

Reconfigurable RISC-V processor design

KEI HONG CHAN

1. Introduction

RISC-V is an instruction set architecture (ISA), which comes from the University of California, Berkeley. Those instructions can be used to build a processor. Since the RISC-V is a free and open ISA, people are not required to apply any patents to design a processor chip. With the aid of the RISC-V, everyone can customize and fully utilize their own chip, without wasting any space or energy for extra function or extra storage that are designed by other companies. Nowadays, many products are required to use microprocessor to do some basic and efficient functions in an embedded system, in general computing and in advanced high-performance computing, companies and researchers are therefore actively involved in the RISC-V community.

In addition, the RISC-V simplifies the processor design. Only a few instructions are required to build a simple processor. So, the RISC-V architecture is suitable for students and someone who is not experts in processor design.

2. Methodology

2.1 Tools introduction

Chisel

Chisel is a hardware design language which can reuse the design from ASIC and FPGA logic design and generate an advanced circuit. In addition, Chisel is embedded in Scala programming language, this can allow the programmer to parameterize the logical circuit and build some complex circuit by using a modern programming language. Based on the above point, the processor can be reconfigured easily by using Chisel, especially rocket processor and BOOM processor.

FIRRTL

FIRRTL is an intermediate representation for digital circuit to transform the program language Scala to another representation. This compiler is used in this project to generate the Verilog file.

Gtkwave

Gtkwave is one of the debuggers to work for analyzing logic rather than using the code in Chisel “printf” to print the logic result out from the Scala program. By generating the .vcd file, users can use the Gtkwave to review the simulation of the chip. Compare the signal view between the code “printf” and the Gtkwave, the Gtkwave is clearer and easier to identify the error design from the chip.

Chipyard

Chipyard is a framework that is used to design a full-system hardware, which can redesign and evaluate the short-term development for SoC. It is a bundle of tools and libraries designed to produce a RISC-V processor from MIMO-mapped peripherals to custom accelerators.

Spike

Spike is a RISC-V ISA Simulator. It can test the compiled RISC-V program for RISC-V processor. The spike also provides the pseudo-operation system (pk) for running the program from the software emulator. Therefore, once the emulator file is generated, the file can be tested by the pseudo-operation system and the RISC-V program to check if the file can function properly.

Vivado

It is a Xilinx FPGA program programmer. Once the generator generates the Verilog from Chisel. It can put into the Vivado to generate the bitstream file for the specified board by a completed IO pin formation. In addition, this can be used for checking the utilization and hierarchy from the generated Verilog to confirm the chip has added all components well in the file for program the FPGA chip.

2.1 Chisel study

To reconfigure the processor, understand how to code Chisel is necessary. By using [1] as a reference for RISC-V processor tutorial and Chisel tutorial, it can help me to understand the structure of RISC-V processor and learn how to use the Chisel to generate a RISC-V processor.

The Chisel is not just used for software simulation, it can also be to generate Verilog code for further development. By using the below code in the top module, the Chisel will generate Verilog through FIRRTL.

```
object target_class extends App {  
  (new ChiselStage).emitVerilog(new target_class())  
}
```

Combined with other programs the Verilog can be used to generate layout for chip design or program into the FPGA.

2.2 Design flow

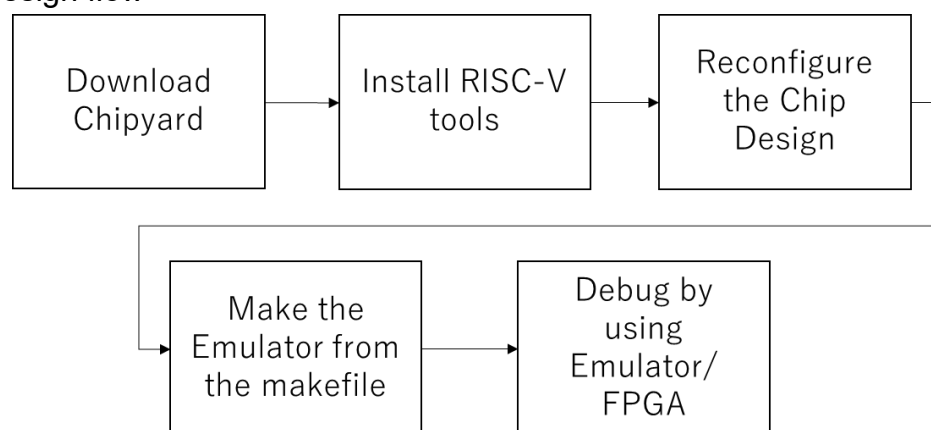


Figure.1: Overview for how the CPU was generated.

The above figure shows the flow to generate a reconfigurable processor in this project.

Step 1: Download the Chipyard

Download the Chipyard from [2] and use it for a CPU generator. It also contains the RISC-V tools installation script.

Step 2: install RISC-V tools

Open the terminal in the Chipyard file and use the command `./scripts/init-submodules-no-riscv-tools.sh` and `./scripts/build-toolchains.sh` to install the tools for RISC-V.

Step 3: Reconfigure the Chip Design

Chipyard is a tool that can reconfigure and generate the processor in a simple way. The coding method to reconfigure the Chipyard processor is at Appendix. A.

In addition, the Chipyard includes some accelerators for the processor, such as Security Hash Algorithm (SHA3) design. It can accelerate the processor when doing the hash algorithm.

Step 4: Make the Emulator from the makefile

To generate the processor for debugging, make the emulator file in the Chipyard is the simplest way. By using the command in Linux Terminal `make CONFIG=configure_class_name`. The emulator will be generated for test ,which can test with the RISC-V compiled file. In addition, using the command `make debug CONFIG=configure_class_name` to generate the debug file for another debug test.

2.3 Method on testing

2.3.1 Debug by using Emulator

After the emulator is generated by the makefile, the emulator file can be tested with the pseudo-operating system (pk) command in Linux terminal (`./designed_cpu pk compiled_RISC-V_file`). If the result is the same as the design output from c program, it is proved the CPU is well designed.

However, if the result is not the same by running the command `./emulator_file-debug pk +verbose -v output.vcd compiled_program 2>&1 | spike-dasm > output.log` to generate a .vcd file to review the processor waveform for more detail information.

By doing a FPGA verification to have more testing on the FPGA to get some result that the software testing cannot get. For example, the SHA3 accelerator cannot be test in software due to the computer is too slow to do this kind of simulation, insert the design into FPGA can used as an external simulator to run the program.

2.3.2 Debug by using FPGA

To make sure the chip that can be run the SHA3 accelector, the design needs to be inserted into the FPGA for further testing.

Due to the time limit, a specified operating system cannot be prepared. Therefore, the RISC-V chip will be programed with another ARM processor into the FPGA and operated with Linux system to check the functionality of the RISC-V processor.

3. Design

In this project the chip design contains a rocket processor with a Security Hashed Algorithm accelerator. By using this design, I can understand how to reconfigure the processor in a high-level coding and how to put the ROCC accelerator into the chip design.

For the rocket chip, it is not reasonable to change the number of ALU into the design, because it needs to change the basic design of the processor structure. Therefore, in this project the number of ALU does not reconfigure. However, if using the BOOM processor design, the number of processor can be reconfigured from the path `/generators/boom/src/main/scala/exu/execution-units/execution-units.scala`.

4. Evaluation

4.1 Utilization

Resource	Estimation	Available	Utilization %
LUT	49628	218600	22.7
LUTRAM	2075	70400	2.95
FF	23761	437200	5.43
BRAM	157.5	545	28.9
DSP	15	900	1.67
IO	326	362	90.06
BUFG	9	32	28.13

Table.1: Utilization for RISC-V processor with sha3 accelerator

BOARD	zc706
PART	xc7z045ffg900-2

Table.2: Board Setup on Vivado

Table.1 shows the utilization on Table.2 mentioned board. The result in table.1 shows LUT usage is 22.7%, on the other word the chip still has space to put more cores into the design.

4.2 Hierarchy

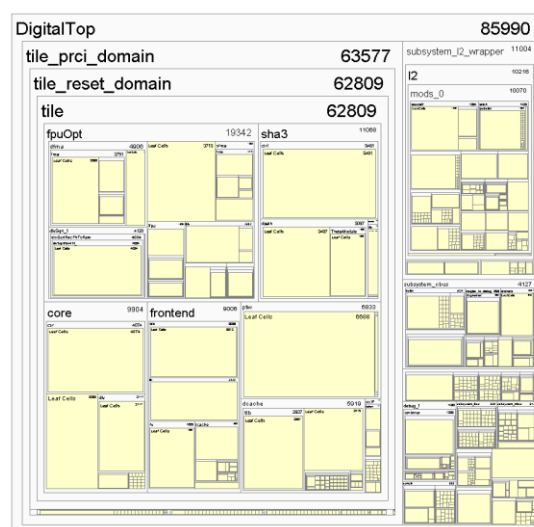
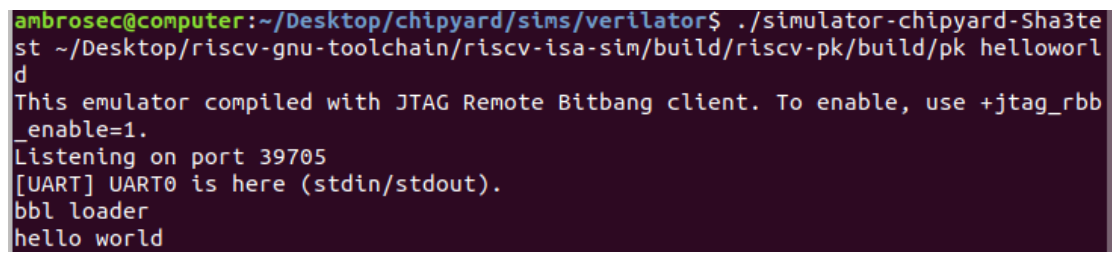


Figure.2: Hierarchy of the chip

Figure.2 shows the sha3 was successfully to add inside to the rocket chip.

4.3 Simulation result



```
ambrosec@computer:~/Desktop/chipyard/sims/verilator$ ./simulator-chipyard-Sha3test ~/Desktop/riscv-gnu-toolchain/riscv-isa-sim/build/riscv-pk/build/pk helloworld
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb_enable=1.
Listening on port 39705
[UART] UART0 is here (stdin/stdout).
bbl loader
hello world
```

Figure.3: the test from the emulator

Figure.3 shows the emulator can run the compiled helloworld program (c program source from Appendix. B)

5. Conclusion

Chipyard is a good tool to help people to have a fast track for generating their own configure chip. However, there are still unstable due to large number of users, and everyone have their situation. For example, the emulator cannot be run with openOCD and GDB in my computer by following the instruction from [4]. Which mean, if the target of RISC-V is making the chip can be generate easily, there are still have a long path to go.

6. Further development

In this project, the RISC-V processor can only with the arm processor using the Linux operation system. To make sure the RISC-V processor can be fully utilized, an operating system for the RISC-V chip is necessary. On the other word, to do the further development it needs more study between RISC-V ISA and operating system to understand how to design a program for the RISC-V processor to get maximum performance of the chip.

Appendix. A

Coding method to reconfigure the Chipyard processor

The below configuration can be added in /generators/chipyard/src/main/scala/config/config.scala				
Main	Type	Size	Code (N for how many core in the processor)	Reference path
Processor	Rocket chip	Small	freechips.rocketchip.subsystem.WithNSmallCores(N)	/generators/rocket-chip/src/main/scala/subsystem/Config.scala
		Med	freechips.rocketchip.subsystem.WithNMedCores(N)	
		Big	freechips.rocketchip.subsystem.WithNBigCores(N)	
	BOOM	Small	boom.common.WithNSmallBooms(N)	/generators/boom/src/main/scala/common/config-mixins.scala
		Medium	boom.common.WithNMediumBooms(N)	
		Large	boom.common.WithNLargeBooms(N)	
		Mega	boom.common.WithNMegaBooms(N)	
		Giga	boom.common.WithNGigaBooms(N)	
Rocc	Sha3	N/A	sha3.WithSha3Accel	/generators/sha3/src/main/scala
	Gemmini	N/A	gemmini.DefaultGemminiConfig	/generators/gemmini/src/main/scala/gemmini
	NVDLA	N/A	nvidia.blocks.dla.WithNVDLA("large", true)	/generators/nvdl/src/main/scala
	HwaCha	N/A	chipyard.config.WithHwachaTest	/generators/hwacha/src/main/scala
L2 Cache	L2 Cache	Customize from the code	freechips.rocketchip.subsystem.WithInclusiveCache(nBanks=number_banks, nWays=n_way_associate, capacityKB=Cache_size)	N/A
example code for config.scala				
<pre>package chipyard import freechips.rocketchip.config.{Config} class Sha3test extends Config(new sha3.WithSha3Accel ++ // add sha 3 acceleator to the chip new freechips.rocketchip.subsystem.WithNBigCores(1) ++ //add rocket chip size big into the chip new chipyard.config.AbstractConfig //using the chipyard basic config into the chip)</pre>				

Table.2: overview of how-to setup the processor in Chipyard configure file

The Table.2 gives the overview of which code can reconfigure the processor size, number of cores, how to add an accelerator into the chip. These codes can be added to the file /generators/chipyard/src/main/scala/config/config.scala. In addition, it can use the code from table bottom as an example to follow to learn how to add a config for the processor.

Step	Command
git clone https://github.com/ucb-bar/rocket-chip.git	Download the rocket chip document
cd rocket-chip	Open the downloaded file
export ROCKETCHIP=`pwd`	
git submodule update --init	update the file
export RISC-V=/path/to/install/RISC-V/toolchain	set the RISC-V tools path
cd \$ROCKETCHIP/emulator	open the emulator file
make CONFIG=config_from_the_above_table	make the emulator file
./generated_file pk helloworld	use the emulator to run the helloworld

Table.3: Step to generate processor [3]

Appendix. B

helloworld.c

```
#include <stdio.h>

int main(void){
printf("hello world\n");
return(0);
}
```

Reference

- [1] RISC-V-FiveStage(2020). Retrieved May 13, 2021, from <https://github.com/PeterAaser/RISC-V-FiveStage>
- [2] Chipyard(2021) . Retrieved May 13, 2021, from <https://github.com/ucb-bar/Chipyard>
- [3]rocket-chip. (2021). Retrieved May 13, 2021, from <https://github.com/chipsalliance/rocket-chip>
- [4] Berkeley Architecture Research ,Chipyard Documentation, (2021)
- [5] AsanoviA, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser,J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao,H., Moreto, M., Ou, A., Patterson, D. A., Richards, B., Schmidt, C., Twigg, S., Vo, H., & Waterman, A. (2016).The Rocket Chip Generator. Berkeley: EECS Department, University of California.
- [6] Schmidt.C Izraelevitz.A A Fast Parameterized SHA3 Accelerator (2013)
- [7] fpga-zynq(2018) Retrieved May 13, 2021, from <https://github.com/ucb-bar/fpga-zynq>
- [8] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avizienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a Scala embedded language. In Proceedings of the 49th Annual Design Automation Conference (DAC '12). Association for Computing Machinery, New York, NY, USA, 1216–1225. DOI:<https://doi.org/10.1145/2228360.2228584>
- [9] Christopher Celio, David A. Patterson, and Krste Asanović. 2015. The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor. Technical Report UCB/EECS-2015-167. EECS Department, University of California, Berkeley. <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html>